PATTERSON & SHERIDAN, LLP
ATTORNEYS AT LAW

3040 Post Oak Blvd, Suite 1500
Houston, TX 77056-6582
TEL 713.623.4844
FAX 713.623.4846

FACSIMILE COVER SHEET

DATE:	April 30, 2008
FILE NO:	IBMK30162
TO:	Examiner: Jennifer To
FAX NO:	571-273-7212
PHONE NO:	571-272-7212
COMPANY:	USPTO
FROM:	Jon Stewart
PAGE(S) with cover:	6
ORIGINAL TO FOLLOW?	☐ YES ☒ NO

APPLICANT INITIATED INTERVIEW REQUEST

U.S. SERIAL NO.:	**10/666,899**
FILING DATE:	**September 18, 2006**
INVENTOR:	**Richard D. Dettinger**
EXAMINER:	**Jennifer To**
GROUP ART UNIT:	**2195**
CONFIDENTIATION NO.:	**8513**

782381_1

MEMO:

FROM: Jon K. Stewart, **Reg. No.** 54,945

TO: **Examiner:** Jennifer To, **Art Unit:** 2195

RE: Proposed Amendments for discussion during Examiner Interview on May 2, 2008, 10am (EST)

SERIAL No. 10/666,899

---

1. (Currently Amended) A computer-implemented method for parallel processing ~~of~~ a sequence of requests received by an application server configured to execute a plurality of threads, comprising:

initiating, by the application server, a first thread ~~primary executing entity~~ configured to (i) perform the sequence of requests received from a user interacting with a web-based application and (ii) to maintain state information specific to the first thread ~~primary executing entity~~, wherein each request, of the sequence, is composed as a uniform resource locater submitted to the web-based application;

initiating, by the application server, a second thread ~~secondary executing entity~~ configured to perform the sequence of requests and to maintain state information specific to the second thread ~~secondary executing entity~~;

performing, sequentially, by the ~~primary executing entity~~ first thread, the sequence of requests; and

performing, sequentially, by the second thread ~~secondary executing entity~~, the sequence of requests, wherein the second thread performs requests, of the sequence, previously performed by the ~~primary executing entity~~ first thread in a specified number of steps behind first thread ~~time-delayed and step-wise fashion~~ while the ~~primary executing entity~~ first thread continues to execute requests, from the sequence of requests, whereby ~~each executing entity~~ the first thread and

second thread each independently maintains ~~its own respective~~ state information ~~independent of, and temporally displaced from, the other executing entity~~ regarding results of each request, of the sequence, performed by the first thread and the second thread, respectively.

2. (Currently Amended) The method of claim 1, ~~further comprising:~~

~~making the~~ wherein the results of performing the sequence of requests by the first thread is ~~performance of the primary executing entity~~ visible to a user; and

~~making the~~ wherein the results of performing the sequence of requests by the second thread is ~~performance of the secondary executing entity~~ transparent to the user.

3. (Currently Amended) The method of claim 1, further comprising:

upon detecting an error in the results from performing, by the first thread, one of the sequence of requests , terminating the first thread ~~primary executing entity~~; and ~~then~~

performing, by the second thread, ~~secondary executing entity~~, at least a portion of the sequence of requests performed by the terminated first thread ~~primary executing entity~~ and not ~~previously~~ yet performed by the second thread ~~secondary executing entity~~.

4. (cancelled) The method of claim 1, wherein the executing entities are threads.

5. (Currently Amended) The method of claim 1, further comprising:

upon completion of the sequence of requests by the first thread, producing ~~output~~ a primary result for the sequence of requests ~~by the primary executing entity~~;

subsequently, upon completion of the sequence of requests by the second thread, producing a secondary result for the sequence of requests ~~output by the secondary executing entity~~;

displaying the ~~output~~ the primary result produced by the first thread ~~primary executing entity~~; and

discarding, without displaying, the secondary result ~~output~~ produced by the second thread ~~secondary executing entity~~.

6. (Currently Amended) The method of claim 1, further comprising, upon detecting ~~encountering~~ an error in the results from performing, by the first thread, one of the sequence of requests ~~primary executing entity~~;

terminating execution of the first thread ~~primary executing entity~~; and

~~recovering from the error by~~ returning a to the user ~~to~~ an indication of the request being ~~handled~~ performed by the first thread resulting in the detected error ~~primary executing entity when being terminated~~

7. (Cancelled) The method of claim 1, further comprising, upon encountering an error by the primary executing entity:

terminating the primary executing entity; and

recovering from the error by returning a user to a request handled by the primary executing entity prior to encountering the error.

The method of claim 1, further comprising:

upon detecting an error in the results from performing, by the first thread, one of the sequence of requests, terminating the first thread ~~primary executing entity~~; and ~~then~~

8. (Currently Amended) The method of claim 1, further comprising, ~~upon encountering an error by the primary executing entity:~~

upon detecting an error in the results from performing, by the first thread, one of the sequence of requests:

terminating the first thread ~~primary executing entity~~; and

recovering from the error by executing remaining requests of the sequence by the second thread up to, but not including, the request resulting in the detected error ~~returning a user to a request within a range of requests between a request being handled by the secondary executing entity and a request being handled by the primary executing entity at the time of encountering the error;~~

~~prompting the user to modify the request resulting in the detected error;~~

~~receiving a modification to the requests;~~

executing the modified request;

executing any remaining requests, of the sequence of requests; and

displaying a final result produced by the second thread from executing (i) the sequence of requests up to, but not including, (ii) the request resulting in the detected error, and (iii) the remaining requests of the sequence, if any.

9. (Original) The method of claim 1, wherein the requests, of the sequence, are time ordered and processed by each of the first thread and the second thread ~~performing executing entity~~ according to the time order.

10. (Cancelled) The method of claim 9, further comprising:

terminating the primary executing entity; and then

performing, by the secondary executing entity, the requests performed by the terminated primary executing entity between a last request handled by the terminated primary executing entity and a last request handled by the secondary executing entity.

11. (Original) A computer-implemented method for parallel processing of requests received by an application server configured to execute a plurality of threads, comprising:

receiving a sequence of ~~user~~ requests from a user;

placing the ~~user~~ sequence of requests on a queue managed by the application server in a time-ordered manner;

performing, by a first thread managed by the application server ~~primary executing entity~~, each ~~current user~~ request, of the sequence, upon being placed on the queue; and

performing, by a second thread managed by the application server ~~secondary executing entity~~, at least a portion of the user requests on the queue step-wise with the first thread ~~primary executing entity~~ and N-requests behind the first thread ~~primary executing entity~~; wherein ~~each of the executing entities~~ maintain their own respective state information the first thread and second thread each independently maintains ~~their own respective~~ state information regarding results of each request, of the sequence, performed by the first thread and the second thread, respectively.

12. (Currently Amended) The method of claim 11, further comprising:

upon detecting ~~encountering~~ an error in the results from performing, by the first thread, one of the sequence of requests ~~primary executing entity~~;

terminating execution of the first thread ~~primary executing entity~~; and

~~recovering from the error by~~ returning ~~a~~ to the user ~~to~~ an indication of the request being ~~handled~~ performed by the first thread resulting in the detected error ~~primary executing entity when being terminated~~.

13. (Cancelled)

14. (Cancelled) The method of claim 11, wherein the executing entities are threads.

15. (Cancelled) The method of claim 11, wherein executing entities are executable code elements of an application.